Les calculatrices sont autorisées

Sujet : page 1 à page 18

PRÉSENTATION DU SYSTÈME

PARTIE 1 - Étude du système ;

PARTIE 2 - Étude de l'authentification de l'utilisateur de la déchetterie ;

PARTIE 3 - Étude du bloc logiciel de traitement du signal intégré au peson ;

PARTIE 4 - Étude de la base de données du SIVOM.

Dossier technique DT 1 à DT 6

Document réponse

- **Vous devez répondre uniquement sur le document réponse**, vous pouvez compléter sur la dernière page si la place vous manque pour une question.
- Le nombre de lignes dessinées pour compléter les algorithmes ou programmes ne présume en rien du nombre de ligne de pseudo-code ou code à écrire.

Les quatre parties sont indépendantes et de proportions assez différentes, c'est pourquoi il est proposé à titre indicatif la répartition du temps suivante :

- partie 1 : 9 % ;
- partie 2 : 31 % ;
- partie 3 : 34 % ;
- partie 4 : 26 %.

Toutefois, il est conseillé au candidat d'aborder et de traiter le sujet dans l'ordre des questions.

La clarté et le soin dans la rédaction des réponses attendues, en particulier dans celles concernant les algorithmes et programmes, seront pris en compte dans la notation finale.

GESTION DE DÉCHETTERIES

PRÉSENTATION DU SYSTÈME

1 - Présentation du système

Ce système permet à une communauté de communes ou SIVOM (**S**yndicat **I**ntercommunal à **VO**cations **M**ultiples) de gérer un ensemble de déchetteries (zones de dépôts de déchets).

Chacune de ces déchetteries doit, par un fonctionnement autonome, permettre les dépôts de déchets des usagers (mesurage de la masse des dépôts de déchets) afin de collecter et, si possible, de permettre de revaloriser les déchets vers un ou des centres de recyclage (voir diagramme de contexte dans le document technique DT1, figure 1).

Ce système est composé d'une ou plusieurs déchetterie(s) associée(s) à un système de gestion centralisée des données (système informatique).

Ce dernier assure la collecte des informations des déchetteries pour un traitement statistique. Il permet également la gestion des usagers (enregistrement, suppression et modification des informations des usagers).

Enfin, il assure la facturation des dépôts des usagers des communes ainsi que des professionnels (artisans).

2 - Sous-système de déchetterie

Le schéma d'une déchetterie est donné dans le document technique DT1, figure 2.

2.1 - Description de la déchetterie

La constitution d'une déchetterie est modélisée par le diagramme de définition de bloc, DT2, figure 3.

2.2 - Rôle de la déchetterie

Son rôle est de permettre aux usagers de déposer leurs déchets dans des bennes spécialisées tout en assurant la traçabilité des dépôts.
Pour chaque opération de dépôt, on souhaite conserver le nom de l'usager, la date, la masse (en kg) du dépôt et le type de déchet déposé.
Les informations nom, date et masse sont mémorisées automatiquement, seul le type de dépôt est validé par le gardien lors de l'opération de pesée des dépôts de l'usager.

2.3 - Fonctionnement de la déchetterie

Le fonctionnement de la déchetterie peut être décrit par la séquence de fonctionnement suivante dans laquelle une condition préalable de fonctionnement est nécessaire (les numéros repères sont présents sur le document technique DT1, figure 2) :

a) Pré-condition de fonctionnement : les feux ⑤ et ⑧ sont rouges.

b) Un usager arrive et s'identifie avec sa carte RFID* au niveau du lecteur RFID ①.

S'il est reconnu et si le nombre d'usagers maximum dans la déchetterie n'est pas atteint et si le pont-bascule n'est pas occupé par un véhicule en entrée ou en sortie, la barrière ③ s'ouvre et l'usager fait monter son véhicule sur le pont-bascule. Le nombre de clients est alors incrémenté de un. Sinon, un message est envoyé sur l'afficheur ② pour informer le client de l'attente nécessaire avant d'entrer sur le pont.

c) L'usager fait peser son véhicule sur le pont-bascule ④. Quand la pesée est terminée, le feu signalétique ⑤ passe au vert autorisant le passage.

d) Quand le client quitte le pont-bascule, le feu ⑤ passe au rouge.

e) L'usager se présente à une benne ⑥ en fonction de ses besoins de dépôts (benne de gravats, benne de déchets verts, benne de carton, …). L'opération e) peut se répéter autant de fois que l'usager a des déchets à déposer dans les différentes bennes.

f) Quand l'usager a terminé ses dépôts, il se présente à la sortie. Il s'identifie avec sa carte RFID au niveau du lecteur ⑦ et attend que le feu ⑧ passe au vert pour accéder au pont-bascule ④.

g) Quand le véhicule monte sur le pont-bascule, le feu ⑧ passe au rouge. Quand la pesée est enregistrée, la barrière ⑨ s'ouvre et le véhicule descend.

h) Quand le pont bascule est vide et après une temporisation, la barrière ⑨ se referme.

i) À la sortie de l'usager de la déchetterie, le nombre de clients est décrémenté de un. Si un usager était en attente en entrée, une identification est lancée. Si les conditions décrites au point b) sont valides alors la barrière d'entrée s'ouvre, autorisant une nouvelle opération de dépose des déchets.

Tout le fonctionnement, décrit ci-dessus, est géré par un ordinateur industriel Ⓐ situé dans le local du gardien.

De manière périodique, cet ordinateur industriel Ⓐ transfère (à son initiative) les données de fonctionnement de la déchetterie vers l'ordinateur bureautique Ⓑ. Ce dernier mémorise les informations dans une base de données appelée « locale ».

*RFID (de l’anglais *radio frequency identification*) : technologie d'identification automatique qui utilise le rayonnement radiofréquence pour identifier les objets porteurs d'étiquettes lorsqu'ils passent à proximité d'un interrogateur. (source : CNRFID - Centre national de référence RFID)

Cette base de données permet ainsi au gardien, via un logiciel spécialisé, de visualiser le fonctionnement journalier de la déchetterie (poids total de la journée, nombre de clients sur les zones de dépose, nombre de visites cumulées, etc.).

Une fois par jour, les données enregistrées dans l'ordinateur de bureautique Ⓑ concernant le fonctionnement de la déchetterie sont envoyées via le réseau ADSL ⑩ à la gestion centralisée du SIVOM.

3 - Sous-système de gestion centralisée

3.1 - Rôle de la gestion centralisée

Son rôle est de garantir la traçabilité en collectant et en mémorisant tous les dépôts de toutes les déchetteries afin d'assurer plusieurs missions :

- permettre le calcul des informations statistiques des dépôts des différentes communes composant le SIVOM ;
- permettre la facturation ;
- permettre l'organisation de la collecte des déchets à destination des centres de recyclage (partie non détaillée dans ce sujet) ;
- fournir à chaque déchetterie la liste, par un fichier, des usagers autorisés à déposer.

3.2 - Fonctionnement de la gestion centralisée

Les données reçues en fin de journée, provenant de toutes les déchetteries, sont enregistrées dans une base de données qui permet d'afficher les informations relatives au fonctionnement (fonctionnement d'une déchetterie ou fonctionnement d'un ensemble de déchetteries) sur des périodes choisies au moment de l'affichage (année, mois, jour).

Deux types de facturation peuvent être établis. Un premier envers les communes, cette facture est établie au prorata des masses déposées par habitant, un second à destination des professionnels (souvent des artisans).

Préambule

On appelle **signature** d'une fonction le nom de la fonction ainsi que l'ensemble des paramètres reçus et éventuellement retournés par cette dernière. Cette signature doit être syntaxiquement correcte vis-à-vis du langage retenu.

On appelle **spécification** d'une fonction la description de l'objectif de la fonction ainsi que la description des paramètres d'entrée et de sortie. Pour les paramètres d'entrée, la description comprend également les contraintes liées à ces paramètres qui assurent le bon fonctionnement de la fonction. La spécification apparaît en commentaire dans le début du corps d'une fonction juste après sa signature.

Exemple « maFonction() »:

Signature : maFonction(paramètre1, paramètre2) : typeDeDonnées

Spécification : cette fonction permet de ….

Entrées : paramètre1, son type (son rôle)

paramètre2, son type (son rôle)

Sortie : son type est typeDeDonnées (son rôle)

PARTIE 1 - Étude du système

Objectif
À partir de la présentation et de la modélisation du système, élaborer l'algorithme permettant de modéliser le fonctionnement partiel d'une déchetterie.

Pour cela, nous allons nous intéresser à la modélisation du comportement du système lors du passage des usagers sur le pont pour entrer dans la déchetterie.

L'applicatif de commande du passage sur le pont bascule est un applicatif « multitâches avec exclusion mutuelle » (l'applicatif complet n'est pas étudié ici) qui contient une tâche qui gère l'entrée dans la déchetterie (étudiée dans le sujet) et une tâche qui gère la sortie de la déchetterie.

Nous allons nous intéresser uniquement à la tâche d'entrée sur le pont bascule qui exécute la fonction **entréeUtilisateur()**.

Vous avez à votre disposition les fonctions suivantes :

authentifier(badge) : booléen
Fonction qui permet de comparer le badge à la liste des badges autorisés.
Entrée : badge, entier (numéro du badge à comparer).
Sortie : autorisation, booléen (vrai si le numéro du badge est présent dans la liste des badges autorisés).

commanderBarriere (numéro , ordre) : néant
Fonction qui commande les barrières.
Entrées : numéro, entier (numéro de barrière) ;
ordre, entier (1 = « ouvrir », 0 = « fermer »).
Sortie : pas de paramètre sortant.

commanderFeu (numéro , couleur) : néant
Fonction qui commande les feux .
Entrées : numéro, entier (numéro du feu) ;
couleur, entier (0 = « vert », 1 = « orange », 2 = « rouge »).
Sortie : pas de paramètre sortant.

détecterPrésencePontPesage ( ) : booléen
Fonction de détection de présence sur le pont.
Entrée : pas de paramètre d'entrée.
Sortie : présence, booléen (vrai ou faux).

incrémenterUsager( ) : néant
Fonction qui permet d'incrémenter le nombre d'usagers dans la déchetterie.
Entrée : pas de paramètre d'entrée.
Sortie : pas de paramètre sortant.

lireBadge( ) : entier
Fonction qui permet de lire le badge présenté par l'usager.
Entrée : pas de paramètre d'entrée.
Sortie : numéro badge, entier.

mémoriser (badge , masse)
Fonction qui permet de mémoriser.
Entrées : badge, entier (numéro du badge lu) ;
masse, réel (masse à mémoriser).
Sortie : pas de paramètre sortant.

peser ( ) : réel
Fonction qui permet de « peser ».
Entrée : pas de paramètre d'entrée.
Sortie : masse, réel (masse de la pesée).

Par ailleurs, les variables suivantes sont disponibles :

- codeRfid (entier contenant le numéro du badge RFID lu) ;
- nbUtilisateur (entier permettant le comptage des usagers) ;
- NbMax (entier constant définissant le nombre maximum de personnes admissibles dans la déchetterie).

Au besoin, vous pouvez créer des variables en les définissant.

Restriction au fonctionnement : nous ne traiterons pas le cas « orange » des feux, ni la détection du passage d'un badge dans le lecteur RFID.

Question 1
Compléter l'algorithme de la fonction **entréeUtilisateur()**, fonction appelée lors d'une demande d'entrée par un utilisateur suite au passage de son badge devant le lecteur RFID d'entrée.

PARTIE 2 - Étude de l'authentification de l'utilisateur de la déchetterie

Dans ce sous-système, nous distinguons deux types différents de communication.

Une première liaison existe entre les lecteurs RFID, le pont-bascule contenant le pont et le peson (organe de pesée), les feux, les barrières, l'afficheur et l'ordinateur industriel. Cette communication est réalisée par une liaison spécialisée de type bus de terrain RS485 «half duplex» avec un protocole particulier décrit dans le document technique DT4.

Une deuxième liaison assure la communication entre l'ordinateur industriel (MOXA) et l'ordinateur de bureautique. Cette communication est une liaison Ethernet TCP/IP.

Objectifs
Identifier les flux d'informations entre les composants de la déchetterie.
Créer une application informatique permettant l'authentification d'un utilisateur avec :
- interrogation du lecteur RFID d'entrée afin d'obtenir le numéro du badge de l'utilisateur ;
- lecture du fichier des badges autorisés ;
- recherche du badge lu dans le fichier.

Pour cela nous allons créer un programme d'authentification à partir de la lecture du badge de l'utilisateur par le lecteur RFID d'entrée et sa recherche dans le fichier des badges autorisés.

La configuration du lecteur RFID est la suivante :
- Identifiant : $50
- Débit : 19 200 bauds
- Bits de données : 8
- Bits de stop : 2
- Parité : aucune

Nous allons nous intéresser à la lecture du numéro de badge fourni par le lecteur RFID d'entrée (protocole donné figure 6 DT4).

Question 2
Justifier que la trame de commande permettant de lire un badge sur le lecteur RFID identifié par le nombre 50 (valeur hexadécimale) s'écrit **`<AA|50|03|25|26|01|51|BB>.`**
La figure 7 (DT4) fournit des rappels relatifs à la fonction Ou Exclusif.

Une fois la trame de commande émise, le lecteur de badge RFID répond en émettant entre autre le numéro d'identifiant.

Question 3
Le document réponse (Question 3) propose un algorithme partiel de gestion de cette réponse.
Il doit permettre d'une part de vérifier l'intégrité de la communication en recalculant l'octet de contrôle (checksum) puis d'autre part, dans le cas d'une communication intègre, il doit déterminer le numéro du badge.

Sachant que la trame réponse est mémorisée dans le tableau trameRéponseBadge[] grâce à l'utilisation de la fonction lectureSurPortSerie(), commenter l'algorithme du document réponse, en vous attardant spécifiquement sur les deux itérations proposées.

Question 4
À l'aide des documentations des fonctions de gestion de la liaison série en Python et en Scilab (figures 8 et 9 DT5), compléter le programme en Python ou en Scilab, permettant d'interroger puis d'afficher le numéro du badge, sachant que la liaison est réalisée par le port série RS485 « COM1 ».

La mise à jour du fichier des badges autorisés dans l'ordinateur industriel est réalisée la nuit lors de la fermeture de la déchetterie, par un applicatif type 2-tiers (client/serveur).

La modélisation des échanges entre un serveur et un client est décrite par la figure 10 DT6.

Question 5
En justifiant votre réponse, donner le type de l'applicatif (client ou serveur), déployé sur l'ordinateur industriel et celui déployé sur l'ordinateur bureautique.

Nous allons nous intéresser maintenant à l'authentification des usagers.

Le lecteur RFID permet de lire le numéro du badge d'un usager. Un fichier contient l'ensemble des numéros des badges autorisés (une version simplifiée de ce fichier est présentée figure 11 DT6). L'identification d'un usager passe alors par la recherche de la présence de son numéro de badge dans le fichier des badges autorisés.

Une première fonction non étudiée ici permet de stocker au préalable l'ensemble des numéros des badges issu du fichier dans un tableau.

En pseudo langage, on considérera que les indices commencent à 1.

Soit l'algorithme de la fonction travail_sur_tableau() suivant :

```
travail_sur_tableau(tab)

Fonction  permettant de ……..
Entrée :        tab[] tableau contenant les valeurs des badges.
Sortie :        ……..

Début
    n←longueur(tab)
    i←2
    Tant que ( i<=n )
        x ← tab[i]
        j ← i-1
        Tant que ( j>0 ET tab[j]>x )
            tab[j+1] ←tab[j]
            j ← j-1
        Fin Tant que
        tab[j+1]← x
        i←i+1
    Fin Tant que
Fin
```

Question 6
Compléter la trace de l'algorithme (tableau d'évolution des variables) pour le tableau d'entrée limité aux cinq premières valeurs du fichier des badges autorisés (figure 11 du document DT6). Donner alors le rôle de l'algorithme proposé ci-avant.

Question 7
Pour la boucle externe, indiquer quelle variable joue le rôle de variant de boucle. Justifier que cette boucle se termine.

Question 8
En justifiant votre réponse, préciser si la classe de la complexité de cet algorithme est linéaire, quadratique, cubique, linéarithmique (quasi linéaire) ou logarithmique.

Nous désirons à présent rechercher la position de l'occurrence du numéro du badge d'un utilisateur dans le fichier des badges autorisés matérialisé ici par le tableau supposé trié tab[].

L'algorithme utilisé est le suivant :

```
recherche(x,tab,m,n)
Fonction  permettant de recherche l'occurrence de x dans tab[]
Entrées :       x : entier , occurrence à rechercher ;
                tab[] : tableau d'entiers ;
                m : entier, borne de recherche minimale ;
                n : entier, borne de recherche maximale.
Sortie :        entier : position de l'occurrence du nombre recherché si trouvé, None sinon.

Début
    si  (m=n)
        alors
                si (tab[m]=x)
                    alors
                            Retourner m
                    sinon
                            Retourner None
                fin si
        sinon
                k<-division entière de (m+n) par 2
                si (tab[k]<x)
                    alors
                            recherche(x,tab,k+1,n)
                    sinon
                            recherche(x,tab,m,k)
                fin si
    fin si
Fin
```

Question 9
Quelle est la particularité de cet algorithme ? Quelle(s) précaution(s) l'utilisateur doit-il prendre pour les valeurs de m et de n lors de l'appel de cette fonction ?

Question 10
Proposer une forme itérative de cet algorithme permettant d'obtenir la position de l'occurrence du badge recherché. Parmi les deux solutions, indiquer alors laquelle utilise le plus de place en pile.

PARTIE 3 - Étude du bloc logiciel de traitement du signal intégré au peson.

Objectif
Valider le traitement numérique du signal effectué lors du pesage d'un véhicule.

Pour cela, nous allons nous focaliser sur :

- le prototypage du traitement en simulation ;
- l'échantillonnage et la nécessité d'un sur-échantillonnage ;
- la réduction du bruit de mesure ;
- l'extraction d'une valeur à retenir pour le pesage.

Vous trouverez l'architecture interne du peson dans le diagramme de définition du bloc peson (figure 4, DT3) ainsi que son diagramme de bloc interne (figure 5, DT3).

3.1 - Modélisation du signal réel

Une conversion analogique/numérique du signal amplifié issu du capteur à base de jauges de déformation a été échantillonné à une fréquence $f_e = 500\ \text{Hz}$. Ce signal de mesure réel (figure S1) est en l'état inexploitable du fait des bruits et des oscillations constatés. De fait, l'amplitude du bruit est supérieure à la précision attendue.

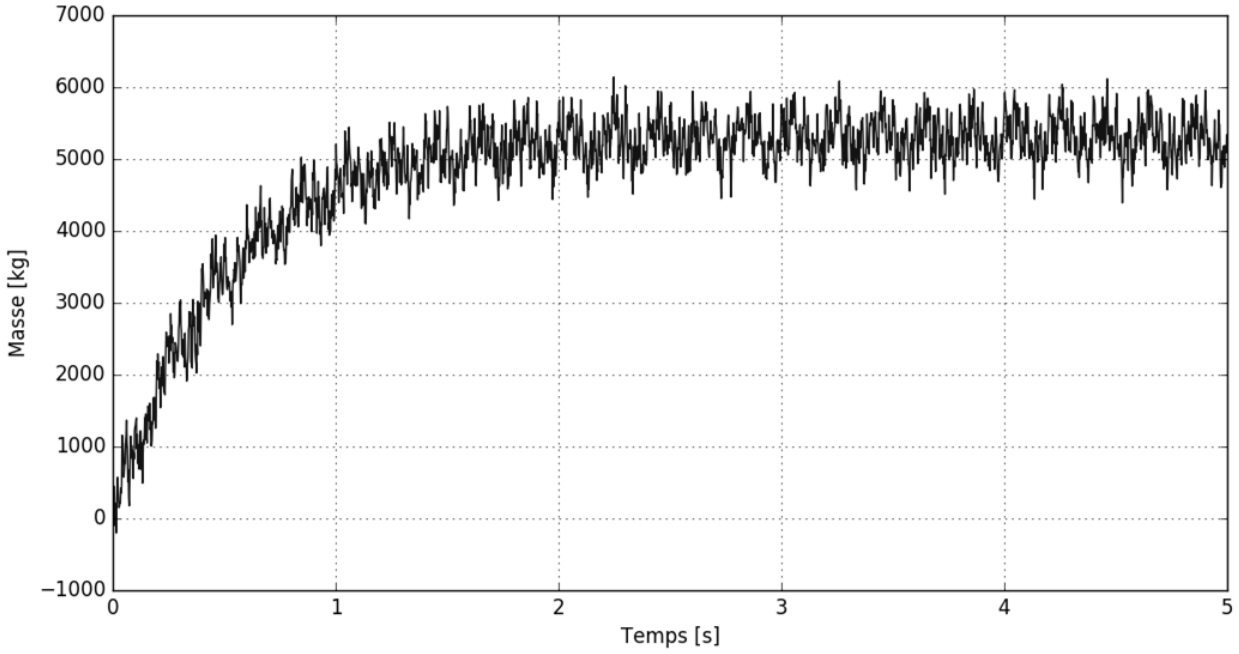


Figure S1 - Exemple de signal mesuré

Nous allons modéliser les principales composantes de ce signal à partir de son spectre, puis nous utiliserons un filtre pour réduire le bruit de mesure. Après avoir validé ce traitement numérique sur le signal réel, nous pourrons extraire la valeur à retenir pour le pesage.

Lorsqu'un signal analogique $x(t)$ est échantillonné avec une période d'échantillonnage T_e, on obtient la suite

$$X_n = x(nT_e)$$

où *n* est un entier.

La suite X_n constitue un signal discret. Lorsque le spectre de $x(t)$ a une fréquence maximale $f_{\max}$, la condition de Nyquist-Shannon permet de s'assurer que toute l'information du signal analogique est présente dans le signal échantillonné : la fréquence d'échantillonnage $f_e = 1/T_e$ doit être supérieure à $2f_{\max}$.

Le calcul du spectre du signal au moyen de la transformée de Fourrier discrète a permis d'obtenir le tracé de la figure S2.

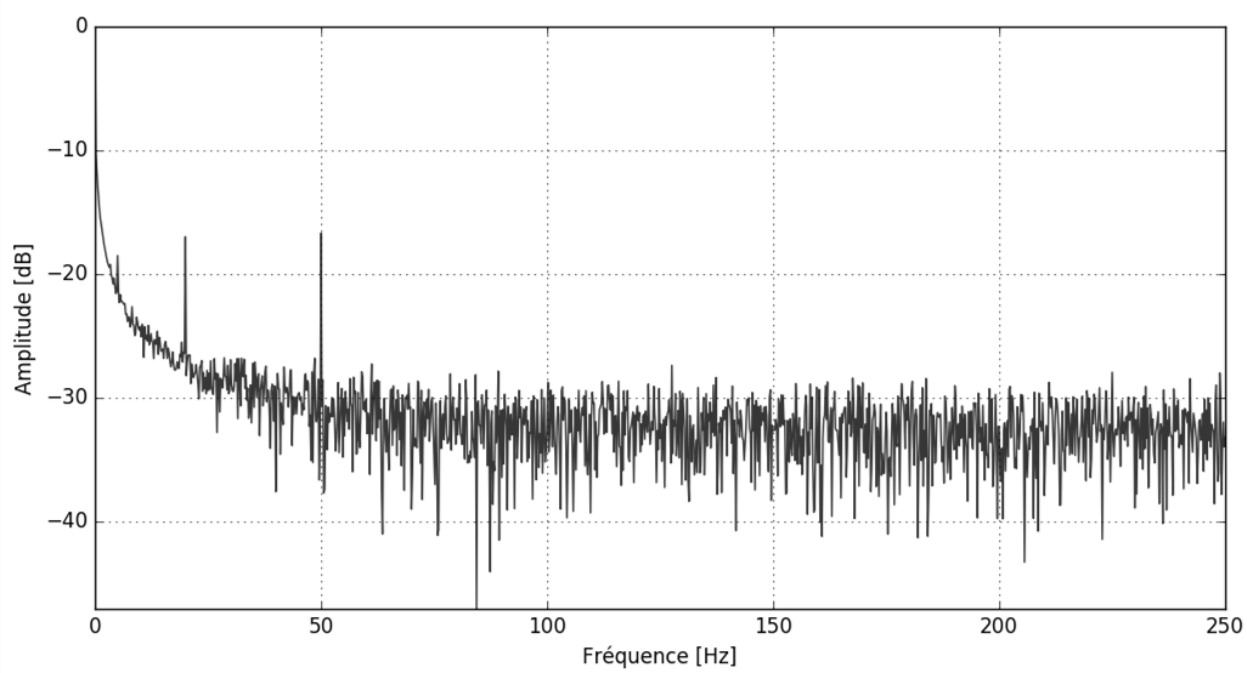


Figure S2 - Spectre du signal échantillonné

Question 11
À partir du spectre du signal échantillonné, donner la valeur de la fréquence minimale d'échantillonnage qui respecterait la condition de Nyquist-Shannon.

Le bruit se manifestant à toutes les fréquences, il est raisonnable de le modéliser par un bruit blanc gaussien. La fréquence d'échantillonnage a été volontairement choisie très largement supérieure aux fréquences utiles du signal. On parle dans ce cas de sur-échantillonnage. Cette condition est nécessaire pour effectuer un filtrage efficace du bruit.

Afin de tester le filtre qui sera utilisé dans l'application, nous allons utiliser un signal simulé représentatif auquel on ajoutera ensuite un bruit gaussien.

Le signal représentatif est déterminé à partir d'une analyse temporelle du signal mesuré d'une part et d'une analyse spectrale d'autre part. Il est la somme :

- d'une réponse indicielle d'un système du premier ordre de constante de temps $\tau = 500$ ms et de valeur finale $x_f = 5\ 320$, soit $s(t) = x_f(1 - e^{(-t/\tau)})$;
- de trois signaux sinusoïdaux modélisant les harmoniques du signal réel :
  - de fréquences respectives : $f_1 = 10$ Hz, $f_2 = 20$ Hz, $f_3 = 50$ Hz ;
  - d'amplitudes respectives : $a_1 = 200$, $a_2 = 200$, $a_3 = 200$;
  - de déphasage : $\varphi_1 = 0$, $\varphi_2 = \frac{\pi}{3}$, $\varphi_3 = \frac{\pi}{5}$, soit pour le premier $a_1 \sin(2.\pi.f_1.t + \varphi_1)$.

Question 12
Justifier les valeurs des paramètres retenus pour les caractéristiques du système du premier ordre.

Question 13
Donner l'expression du signal $x_1(t)$ ainsi défini.

Question 14
Écrire la fonction signal() prenant en argument la variable t et renvoyant la valeur du signal $x_1(t)$.

Finalement, pour construire le signal modélisant la mesure issue du capteur, on ajoute à x_1 un signal appelé bruit blanc. Il s'agit d'un signal aléatoire dont les composantes spectrales couvrent une large bande fréquentielle.

On parle de bruit gaussien lorsque la densité de probabilité de cette variable est la loi gaussienne (ou loi normale).

En choisissant une loi gaussienne centrée, la densité de probabilité est :

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\left(-\frac{x^2}{2\sigma^2}\right)}.$$

La probabilité pour que le bruit soit compris entre x et $x + dx$ est $p(x)dx$. Les signaux délivrés par les instruments de mesure comportent un bruit qui peut être généralement considéré comme gaussien. Cette hypothèse permet de définir l'écart-type σ comme l'incertitude type due aux erreurs aléatoires.

On dispose d'un générateur de nombre pseudo-aléatoire délivrant des nombres dans l'intervalle [0,1] avec une densité de probabilité uniforme. La génération de nombres aléatoires vérifiant la distribution gaussienne se fait alors en utilisant le théorème suivant : si U_1 et U_2 sont deux variables aléatoires uniformes sur l'intervalle [0,1] alors

$$\sigma\sqrt{-2\ln(U_1)}\cos(2\pi U_2)$$

est une variable aléatoire vérifiant la loi gaussienne centrée.

Question 15
Compléter la fonction bruitGauss(), prenant en argument l'écart-type sigma et renvoyant un nombre aléatoire vérifiant la distribution gaussienne.

Pour compléter la synthèse du signal, on ajoute donc le bruit gaussien. Le signal bruité x_2 est donné par $x_2 = x_1 + b$. Le temps de simulation est fixé à $\mathrm{T} = 5\ \mathrm{s}$. On échantillonne à une fréquence $\mathrm{f_e} = 500\ \mathrm{Hz}$, l'écart-type sigma est fixé à $\sigma = 0,3$ et son amplitude à `ampbruit` = 500.

Question 16
Compléter le programme permettant d'obtenir le tableau des valeurs du signal de test $x_2(t)$ et permettant de présenter la réponse temporelle du signal modélisé.

3.2 - Réalisation de la fonction de convolution

On dispose maintenant d'un signal simulé qui modélise le signal réel, nous allons donc tester un filtre pour réduire le niveau de bruit.
Un filtre numérique à réponse impulsionnelle finie réalise le calcul d'un signal discret de sortie y_n en fonction du signal d'entrée x_n de la manière suivante :

$$y_n = \sum_{k=0}^{N-1} h_k x_{n-k} \ .$$

La valeur de la sortie à l'instant n est donc une combinaison linéaire des valeurs de l'entrée à l'instant n et aux $N-1$ instants antérieurs. Si les valeurs de l'entrée ne sont disponibles qu'à partir de l'indice 0, la première valeur de la sortie est y_{N-1}, puisqu'il faut au moins n éléments en entrée pour faire le calcul. Les coefficients h_0, $h_1 ... h_{N-1}$ constituent la réponse impulsionnelle du filtre. L'opération ainsi définie est un produit de convolution.

Question 17
Compléter la fonction `convolution()` qui prend en argument un tableau de coefficients `H` et un tableau de valeurs `x` et qui renvoie le tableau des valeurs de la sortie `y` obtenue par application du produit de convolution.

3.3 - Obtention des coefficients du filtre numérique

Pour réduire le bruit, on utilise un filtre gaussien dont la réponse impulsionnelle est une gaussienne définie par :

$$f(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{\left(-\frac{t^2}{2\sigma^2}\right)} \ .$$

On choisit une longueur de réponse impulsionnelle impaire $N = 2P+1$, ce qui permet de centrer la gaussienne sur l'indice $k = P$. La réponse impulsionnelle est alors donnée par :

$$h_k = f(k-P).$$

Si P est donné, on choisit σ pour avoir une valeur ε très faible sur les bords :

$$\sigma = \frac{P}{\sqrt{-2\ln\varepsilon}}.$$

La réponse impulsionnelle est ensuite normalisée pour que la somme de ses coefficients soit égale à 1. On choisit les valeurs suivantes : $P = 200,\ \varepsilon = 0,0001$ (figure S3).

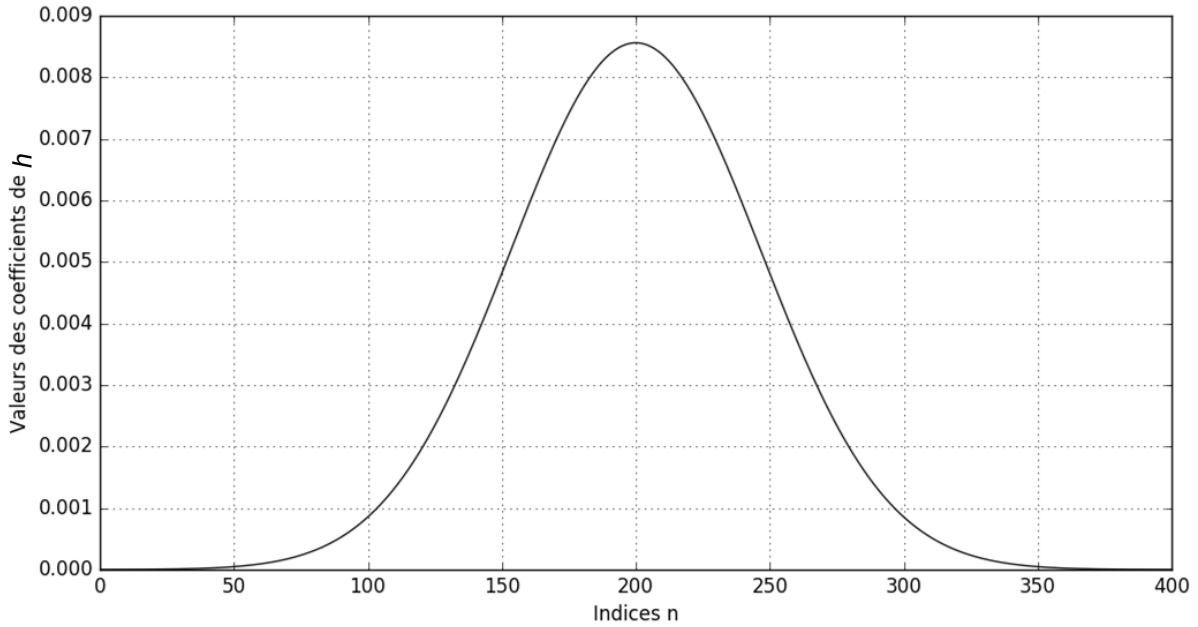


Figure S3 - Réponse impulsionnelle du filtre gaussien

Question 18

Écrire le programme permettant de calculer le tableau de données représentant la réponse impulsionnelle normalisée h.

3.4 - Obtention du signal filtré

Finalement, le signal filtré s'obtient par la commande

```
Signal_Filtre=convolution(H,X)
```

avec `H`, le filtre gaussien et `X`, le signal réel à filtrer.

L'application de ce filtrage a permis d'obtenir sur le tracé du signal bruité, celui du signal filtré. Les deux sont représentés sur la figure S4.

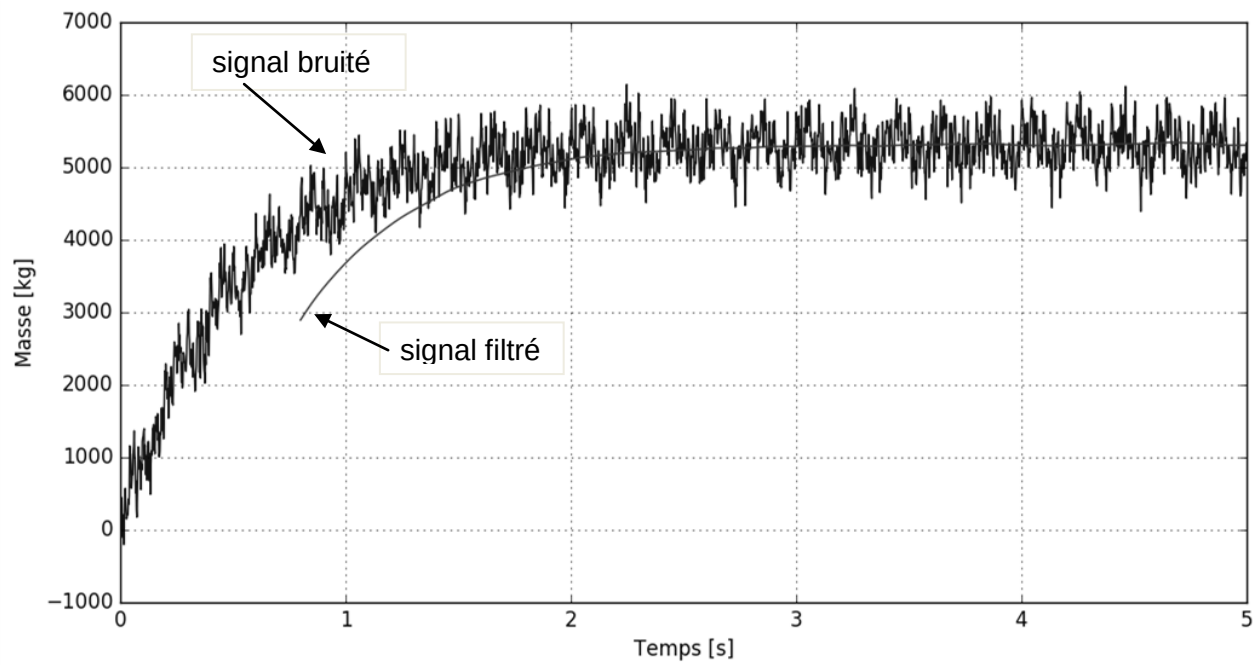


Figure S4 - Filtrage gaussien du signal bruité

Question 19
Que pensez-vous du signal ainsi obtenu ?
Permet-il d'avoir une image de la masse du chargement ?

PARTIE 4 - Étude de la base de données du SIVOM

Objectif
Identifier les transferts d'informations entre les sous-systèmes et étudier les flux de données (emmagasinées dans chacune des déchetteries) et leur transport, permettant une gestion centralisée des informations reçues au niveau du SIVOM sous forme d'une base de données ainsi que la présentation de ces informations.

Pour cela, nous allons nous intéresser :

- aux échanges d'informations entre le SIVOM et une des déchetteries ;
- à la conception de la base de données du SIVOM ;
- au mécanisme enregistrement/lecture d'informations dans cette base ;
- à l'exploitation des données de la base ;
- à la présentation des données.

Question 20
A partir de la description du système, rappeler les informations transitant entre le SIVOM et une des déchetteries ainsi que leur fréquence.

Question 21
Nous allons nous intéresser à la base de données centralisée (SIVOM) dont vous trouverez une vue partielle dans **le document réponse** (question 21). Nous voulons mémoriser les dépôts caractérisés par leur date, leur type de dépôt (fer, déchet vert, carton, etc.) et leur masse.

Ces dépôts sont réalisés dans les différentes déchetteries par des usagers (table Utilisateurs).
Ces déchetteries sont caractérisées par un identifiant, un nom, une rue, une commune, un numéro de téléphone, le nom du responsable et l'heure de synchronisation des données.

Compléter la base de données centralisée du SIVOM, précitée, en donnant les attributs de chaque entité « dépôts » et « déchetteries » et en spécifiant les identifiants (ou clé primaire) de chaque table ainsi que les clés étrangères.

Nous allons nous intéresser ensuite à l'applicatif « traitement des données » qui permet :

- la gestion des utilisateurs (ajout, suppression ou modification des paramètres) ;
- la réalisation de statistiques sur les dépôts des utilisateurs ;
- l'impression de factures envers les professionnels.

Des contenus partiels des tables sont donnés dans la suite (*toutes les données présentes dans les tables ci-après ne sont que spéculations et ne reflètent pas des personnes réelles*).

Contenu partiel de la table : UTILISATEURS

UTILISATEURS								
ID utili-sateur	**Commune_ ID commune**	**NOM**	**Prénom**	**Rue**	**Tel**	**Mail**	**Auto--risation**	**artisan**
100	12	DUPOND	Jean	De la poste	0 123 456 788	Jean.dupond @mail.fr	oui	non
101	78	DURAND	Daniel	De la mairie	0 109 876 543	Daniel.durand @mail.fr	non	non
120	12	PIERRE	Daniel	Du centre	0 109 876 589	Daniel.pierre @mail.fr	oui	oui
50	12	LAPIERRE	Michel	De la gare	0 202 030 304	Michel.lapierre @mail.fr	non	oui

Contenu partiel de la table : COMMUNES

COMMUNES			
IDCommune	**Code Postal**	**Nom**	**Tel Mairie**
12	89100	SENS	0123456789
21	89700	TONNERRE	0612121212
78	89300	JOIGNY	0198765432
87	89000	AUXERRE	0723232323

Question 22

Nous souhaitons créer un courrier électronique à destination des artisans afin de leur préciser des horaires d'ouverture spécifiques pour les professionnels.

Nous souhaitons ainsi obtenir les noms et adresses mail de tous les artisans.

Donner la requête SQL et le résultat.

Question 23
Nous voulons ajouter dans la base de données un nouveau déposant : monsieur DUPONT Christian, habitant rue de la poste à VILLENEUVE SUR YONNE (code postal 89500). M Dupont n'est pas un artisan, mais il est autorisé à déposer des déchets.

Donner la(es) condition(s) (appelée(s) contrainte(s) référentielle(s)) portant sur les tables de la base de données, qui permettent d'ajouter un nouvel habitant dans la table « UTILISATEURS».
Donner la requête SQL permettant d'ajouter M DUPONT dans la table en prenant comme IDCommune la valeur 15 pour la commune de VILLENEUVE SUR YONNE.

Question 24
Afin de créer un courrier par mail d'information sur la mise en place de nouveaux badges, nous souhaitons obtenir la liste alphabétique des habitants de la commune de SENS qui sont autorisés à déposer.

Donner la requête SQL et le résultat.

Nous voulons estimer la valorisation des déchets de type « vert » par biomasse. Pour chaque tonne de déchets de type « vert », on estime les frais de collecte à 50 € et la revente à 60 €.

Remarque 1 : on considèrera pour la question suivante que les attributs « type de dépôt » et « date » existent bien dans la table « DEPOTS ».

Remarque 2 : les dates sont gérées sous la forme d'un nombre entier représentant l'heure et la date à la seconde près depuis le 1/1/1970. Vous pourrez utiliser la fonction TIMESTAMP('22/02/2015') qui renvoie l'entier correspondant.

Question 25
En utilisant les fonctions d'agrégation (sum() et count()) et les alias en SQL, proposer une requête SQL permettant de déterminer le nombre de dépôt de type « vert » (alias nbDepotVert), la masse totale déposée (alias masseTotaleVert) ainsi que le budget en recette de la valorisation de ces déchets (alias recette) pour chaque déchetterie sur l'année 2015.

Fin

Dossier technique

DT1

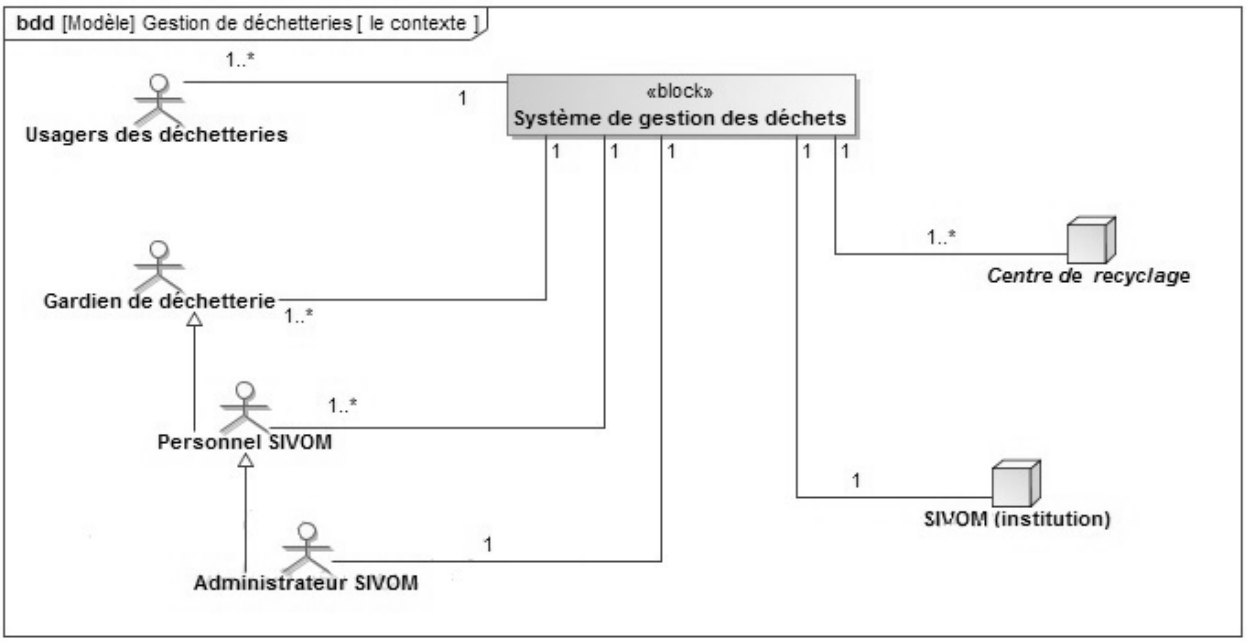


Figure 1 - Modélisation SysML - diagramme de contexte du système

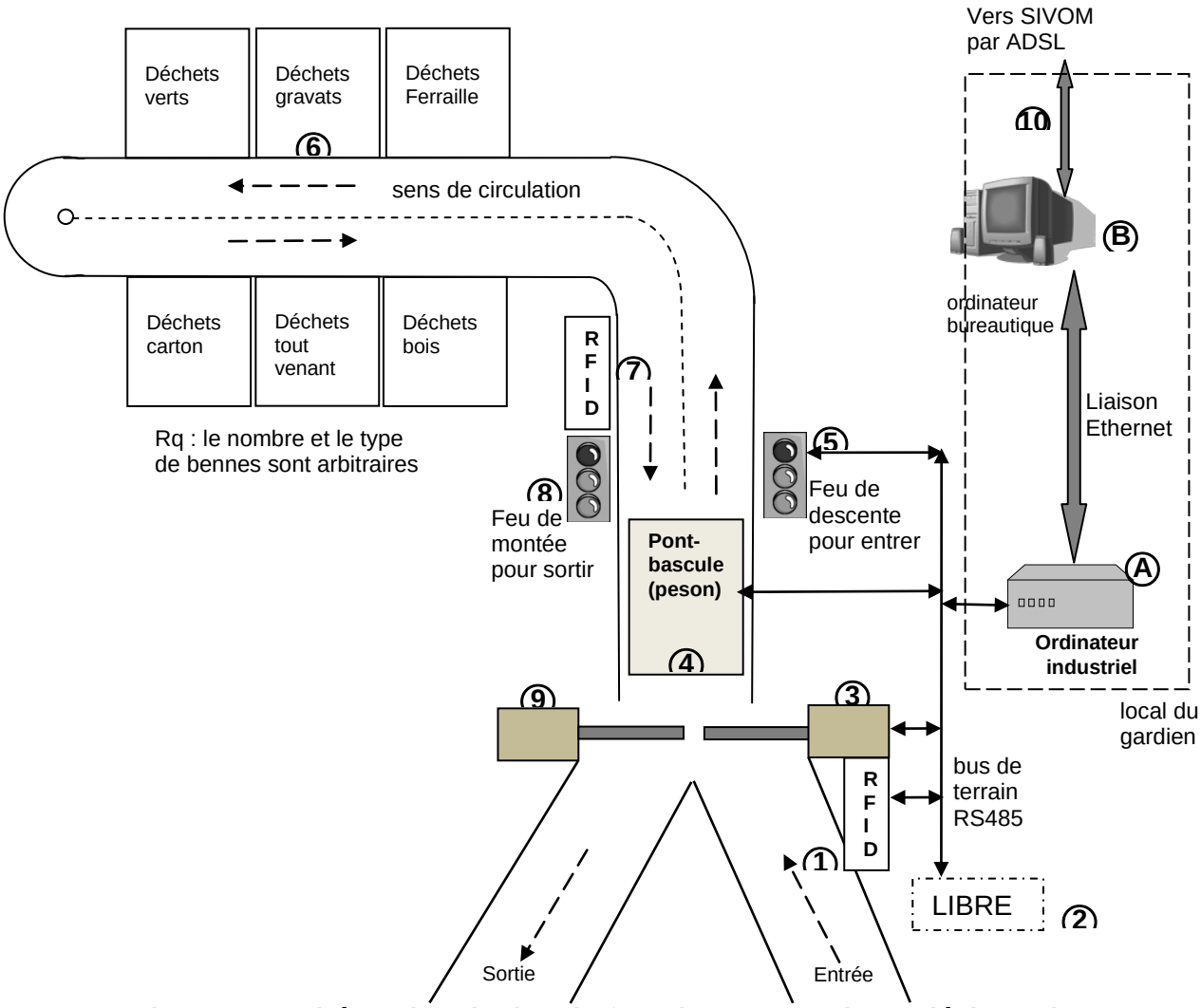


Figure 2 - Schéma de principe du fonctionnement d'une déchetterie

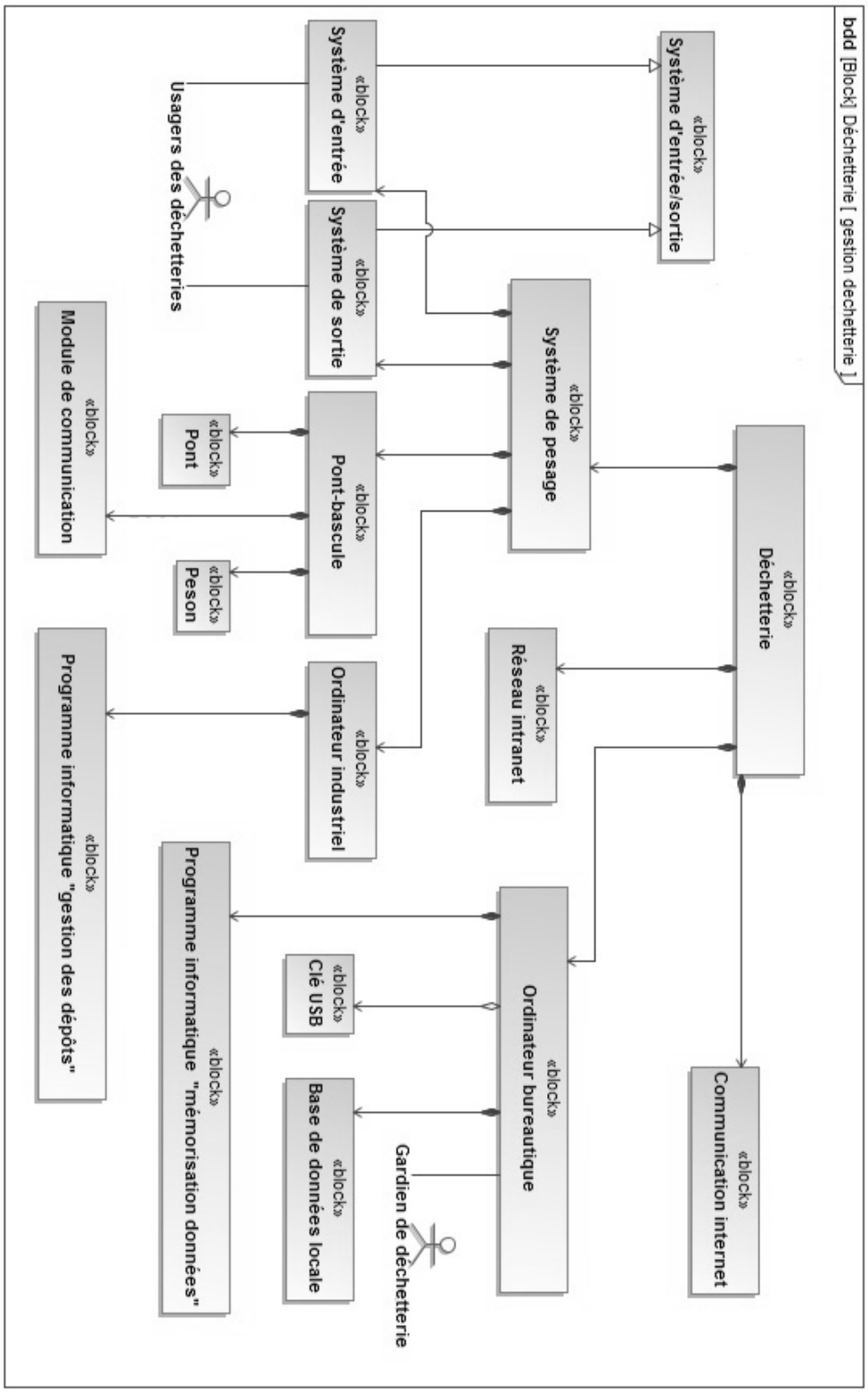


Figure 3 - Modélisation SysML - diagramme de définition de bloc d'une déchetterie

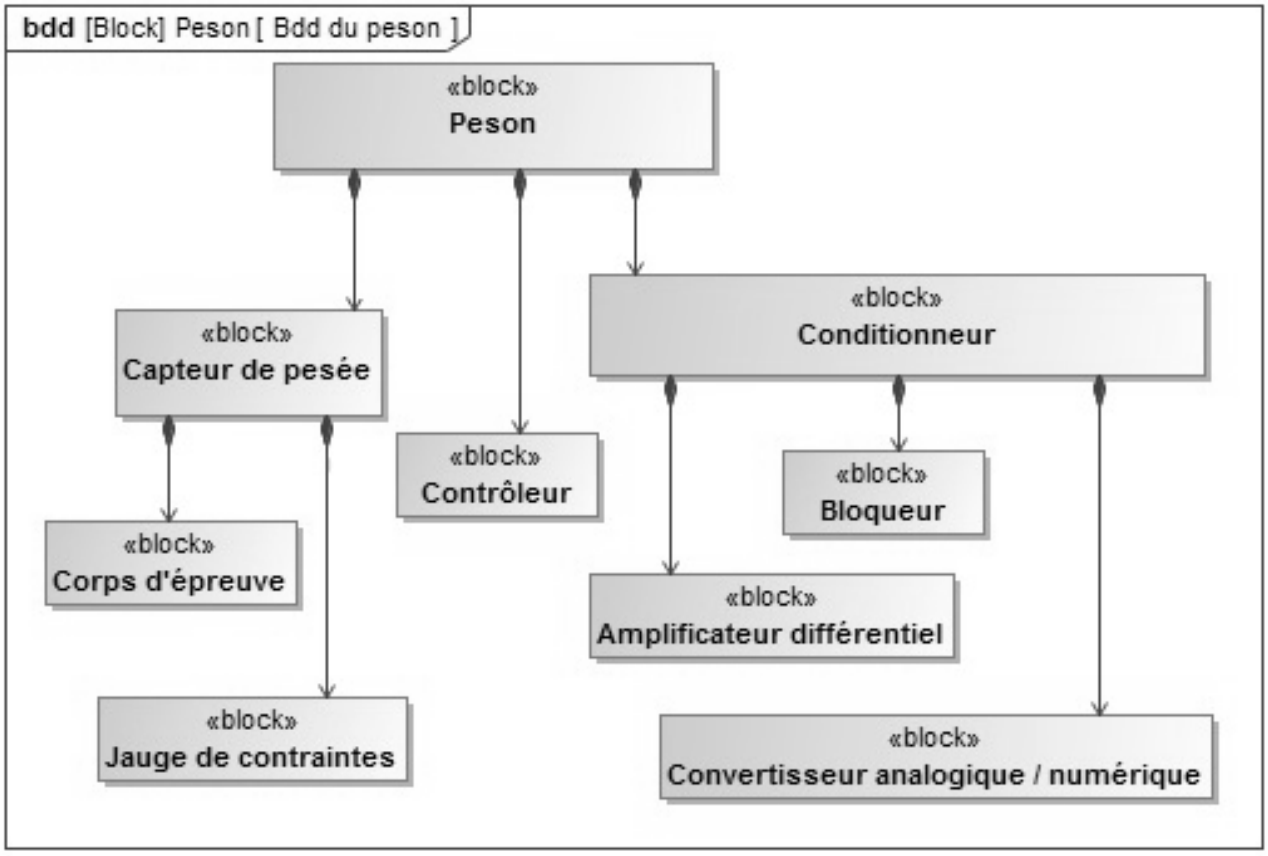


Figure 4 - Modélisation SysML - diagramme de définition de bloc du peson

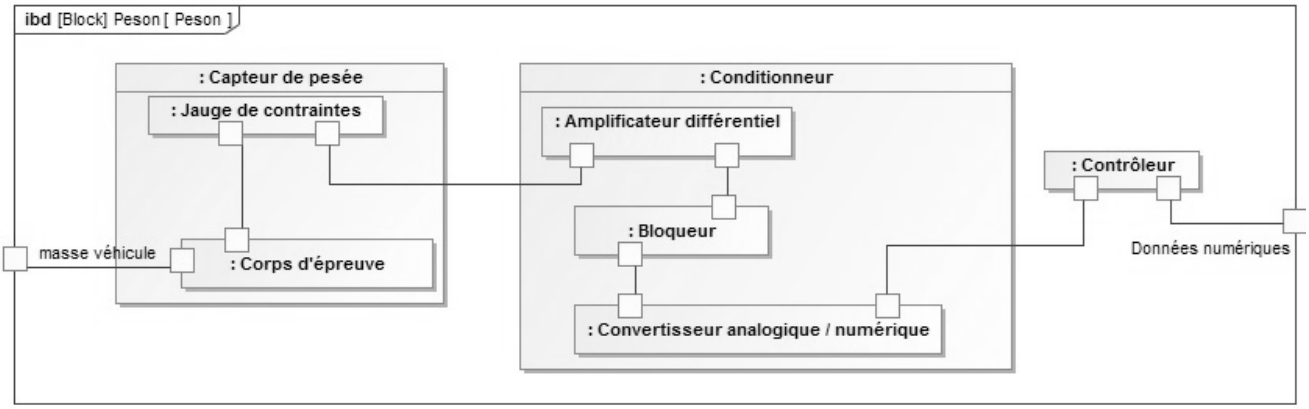
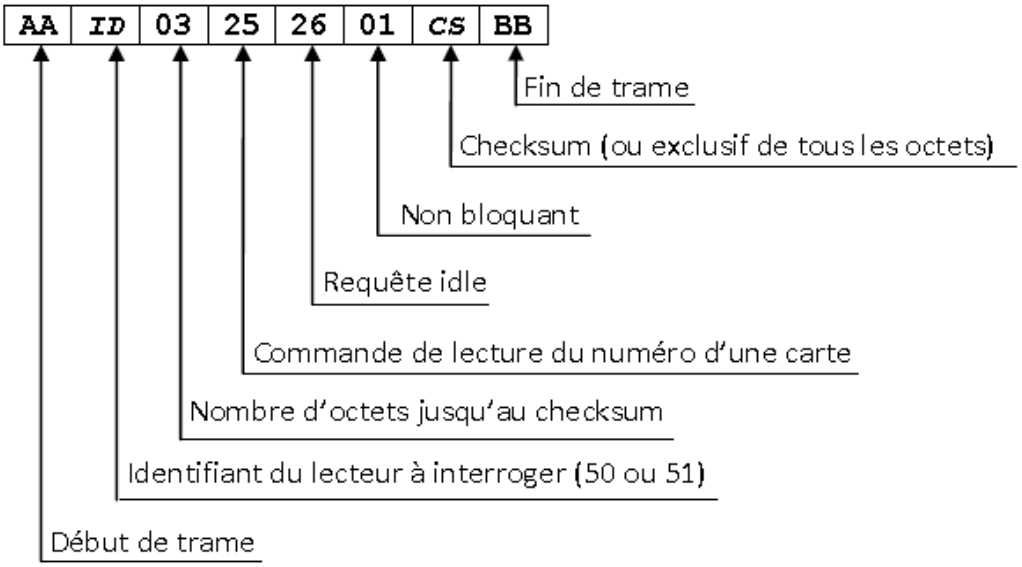


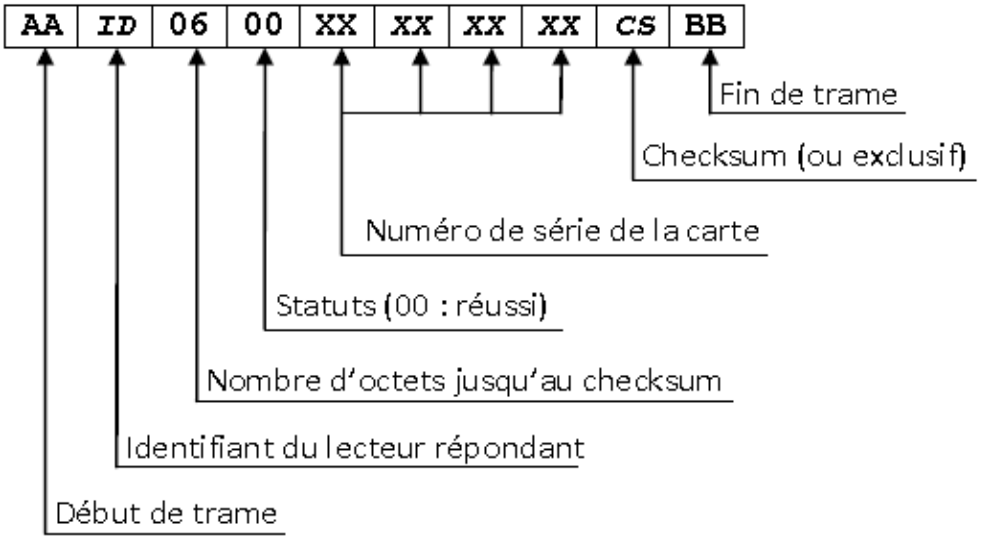
Figure 5 - Modélisation SysML - diagramme interne du bloc peson

DT4

Trame de commande de lecture carte (valeur hexadécimale, 1 octet par case) :



Trame de réponse du lecteur RFID (valeur hexadécimale, 1 octet par case, XX XX XX XX : octet de poids fort du numéro du badge en première position codage en big endian)



Remarque pour les 2 trames : les octets de début de trame (hexa AA) et de fin de trame (hexa BB) ne sont pas pris en compte dans le calcul de l'octet de contrôle (checksum).

Figure 6 - Protocole lecteur RFID sur liaison série RS485

Table de vérité

a	b	a Ouex b
0	0	0
0	1	1
1	0	1
1	1	0

Entre deux octets

A	10101010	$AA
B	11001100	$CC
A Ouex B	01100110	$66

Codage en python
C=A^B

Codage en Scilab
C = bitxor(A,B)

Figure 7 - Rappel Ou Exclusif

librairie serial

- Configuration de la liaison série

```
ser=serial.Serial ( port , baudrate, bytesize, parity='N',stopbits )
```

renvoie un manipulateur sur l'objet série.

- Ouverture du canal de transmission

```
ser.open()
```

- Test d'ouverture du canal de transmission renvoie un booléen vrai si ouverture réalisée

```
ser.isOpen()
```

- Lecture de 100 octets en provenance de la liaison série

```
s = ser.read(100)
```

- Ecriture de données sous forme d'octet passé en paramètre

```
ser.write(….)
```

- Fermeture du canal de transmission

```
ser.close()
```

Figure 8 - Pyserial : fonctions de la bibliothèque liaison série en Python

Serial Communication Toolbox for Scilab

- Configuration de la liaison série et ouverture du canal de transmission

```
h= openserial ( port , " baudrate,  parity='N',  bytesize, stopbits ") ;
```

```
// renvoie   h, un manipulateur sur l'objet série.
```

- Lecture de 100 octets en provenance de la liaison série

```
buf= readserial (  h , 100)
// avec h manipulateur sur l'objet série et 100 nombre d'octets à lire
//     buf  manipulateur sur l'objet contenant les données lues
```

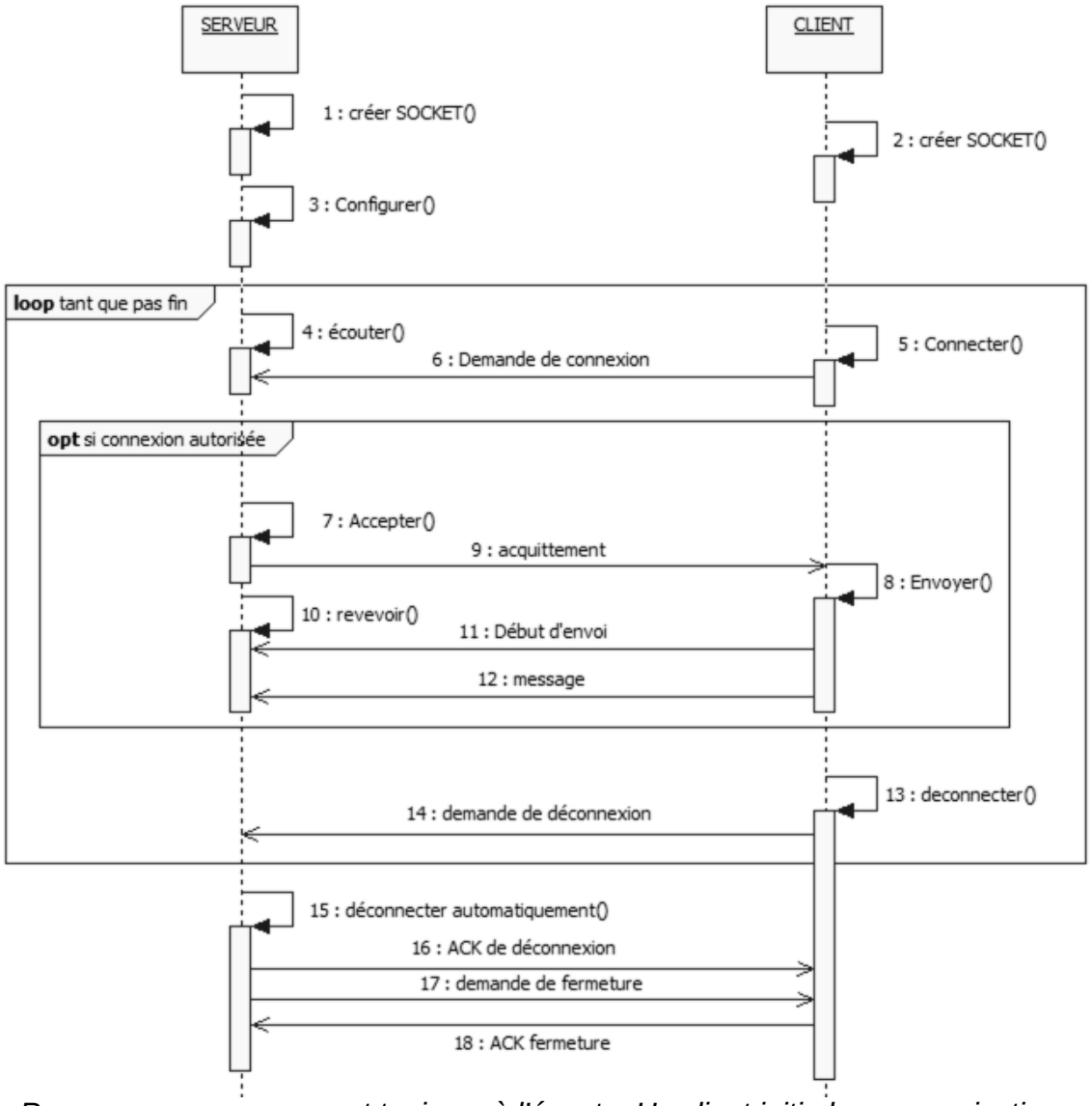
- Ecriture de données

```
result = writeserial(   h ,   buff )
//avec h manipulateur sur l'objet série et buff manipulateur sur l'objet contenant les
données à  envoyer sur le port série.
```

- Fermeture du canal de transmission

```
result = closeserial ( h )
```

Figure 9 - Fonctions de la bibliothèque liaison série en Scilab



Remarque : un serveur est toujours à l'écoute. Un client initie la communication.

Figure 10 - Modélisation des échanges TCP/IP entre un serveur et un client (diagramme de séquence)

12020501
12020508
12020506
12020502
12020504
12020509
12020503
12020505
12020507
12050605
12050601
……..

Figure 11 - Contenu du fichier des badges autorisés sur l'ordinateur industriel (nom du fichier : fichier_badges_auto.txt)

Document réponse

Le langage est laissé libre, au choix du candidat : Python ou Scilab.

Le choix est identique pour toutes les questions du sujet.

Vous entourerez sur le document réponse votre choix de langage :

exemples :

Question x en Python **ou** **Question x en Scilab**

Question 1
Algorithme à compléter

```
Fonction entréeUtilisateur()

Début
        Tant que (badge non présent)   faire

                        afficherMessage(''Passez votre badge dans le lecteur'') ;

        Fin Tant que

        codeRfid ← lireBadge( ) ;

        si (  (authentifier(codeRfid) ) ET

                        ( nbUtilisateur < NbMAX) ET _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ )
           alors
              _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
               _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

           sinon
               AfficherMessage(''veuillez attendre et/ou vous authentifier'')

        fin si
Fin
```

Question 2

Justifier la trame de commande **<AA|50|03|25|26|01|51|BB>**

Question 3

Algorithme à commenter (utiliser les zones de commentaires)

```
/*En pseudo langage, on considérera que les indices commencent à 1.*/

var :
        trameLectureBadge[AA,50,03,25,26,01,51,BB] ;
        trameRéponseBadge[10] ;
        checksum=00 ;       /*Hexadécimal*/
        num_badge=0;

Début programme
|
|       canal ← configurer_Communication_avec_lecteur_RFID() ;
|       ouverture du canal de transmission(canal) ;
|
|       /*envoi trame requête*/
|       ecrireSurPortSérie  (trameLectureBadge , portSérie ) ;
|
|       /*réception trame réponse*/
|       trameRéponseBadge ← lectureSurPortSérie(10) ;
|
|       /*calcul du ckecksum*/
|       Nb_octet← trameRéponseBadge[3] ;  /*_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ */
|       Pour  i  de 2 à Nb_octet+2 inclus  faire /* _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _*/
|       |
|       |       checksum←checksum OUEX trameRéponseBadge[i]      ;
|       |       /* _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _   */
|       fin pour
|       /* explication itération Pour   _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|       _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|       _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|       _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|       _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|       _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _*/
|
|       /*suite algorithme page suivante */
|
```

(suite Question 3)

```
/* Vérification du checksum */
si (checksum= trameRéponseBadge[3+Nb_octet] )
        /* _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ */
 alors
    /*  calcul du numéro de badge */
      Pour   i de 5  à  nb_octet+2  inclus  faire
          /* _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ */

           j←nb_octet+2-i   ;
            /* _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ */
           num_badge←num_badge +( trameRéponseBadge[i]*16puissance(j) )  ;
           /* _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
           _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
           _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
           _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _   /*
      fin pour
    /* explication itération Pour  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _*/

 sinon afficher (’’votre badge n’est pas reconnu ’’ ) ;

fin si

Fin programme
```

Question 4 en Python

Programme en Python à compléter

```
Import serial
#les variables
Demande= _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

CS=0
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
#configuration de la liaison
Ser=serial.Serial( _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _)
Ser.open()

#envoi de la requête
Ser.write(Demande)

#réception de la réponse
réponse= Ser.read(10)

#calcul du checksum
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

#calcul du numéro de badge reçu si le checksum correspond
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

Question 4 en Scilab

Programme en Scilab à compléter

```
// les variables
Demande= _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
CS=0
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
// configuration de la liaison
Ser=openserial(_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _)

/ /envoi de la requête
writeserial( _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _, Demande)
sleep(1000);
// réception de la réponse
réponse= readserial( _ _ _ ,10)

// calcul du checksum
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

// calcul du numéro de badge reçu si le checksum correspond
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

Question 5

Applicatif déployé sur ordinateur industriel : _

_ _

Applicatif déployé sur ordinateur bureautique : _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

_ _

Question 6

i	Nb de boucles Tant que	Liste à la fin de la boucle Pour				
État initial		**12020501**	**12020508**	**12020506**	**12020502**	**12020504**
2	**1**	**12020501**	**12020506**	**12020508**	**12020502**	**12020504**
3		**12020501**				
4						
5						

Rôle de l'algorithme : _

_ _

_ _

_ _ _ _ _ _ _ _ _ _ _ _ _ _

Question 7

Variable : _

Justification **:** _

_ _

_ _

_ _

_ _

Question 8

Classe de complexité :

_ _

_ _

_ _

Question 9

Particularité de l’algorithme :

Précaution(s) sur m et n :

Question 10

Solution itérative :

Place dans la pile :

Question 11

Valeur de la fréquence minimale d’échantillonnage :

Question 12

Justifier les valeurs des paramètres retenus :

_ _
_ _
_ _
_ _
_ _
_ _

Question 13

$x_1(t) =$ _
_ _
_ _
_ _
_ _

Question 14 en Python

Écrire la fonction `signal(t)`

```
import numpy
def signal(t):
     return_( _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
     _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
     _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
     _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _)
```

Question 14 en Scilab

```
function  x1  = signal(t)
     x1 =_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
     _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
     _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
     _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
endfunction
```

Question 15 en Python

```
import random
def bruitGauss(sigma):
    U1 = random.random()
    U2 = random.random()
     _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    return (_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _)
```

Question 15 en Scilab

```
function bruit  = bruitGauss(sigma)
     rand(”normal”)
     U1 = rand
     U2 = rand
     bruit =_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
endfunction
```

Question 16 en Python

Compléter le script Python

```
# figure du signal de test
from matplotlib.pyplot import *
fe = 500
T = 5.0
N =_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
te = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
x2 = numpy.zeros(N)
t = numpy.zeros(N)
ampbruit = 500
sigma = 0.3
for k _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _:
    t[k] = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
           _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    X2[k]= _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
           _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
figure(figsize=(12,6))
plot(t,x2)
xlabel('temps [s]')
ylabel('signal de test x2 [kg]')
grid()
title('le signal de test')
```

Question 16 en Scilab

Compléter le script Scilab

```
// figure du signal de test
fe = 500
T = 5.0
N = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
te = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
x2 = zeros(N)
t = zeros(N)
ampbruit = 500
sigma = 0.3
for k _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    t(k) = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
           _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

    x2(k) = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
           _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
end
fig = figure()
plot(t,x2)
xlabel("temps [s]")
ylabel("signal de test x2 [kg]")
xgrid()
title("le signal de test")
```

Question 17 en Python

```
# fonction convolution
def convolution(H,x):
    N = H.size
    nx = x.size
    ny =  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    y = numpy.zeros(ny)
    for n in range(ny):
        for k in range(_ _ _ _ _ _ _ _ ):
            y[n]+=  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
                    _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    return y
```

Question 17 en Scilab

```
function y=convolution(H,x)
    [l,N] = size(H)
    [l,nx] = size(x)
    ny = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    y = zeros(ny)
    for n = 1:ny
        for k = _ _ _ _ _ _ _ _ _ __
            y(n) = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
                   _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        end
    end
endfunction
```

Question 18 en Python

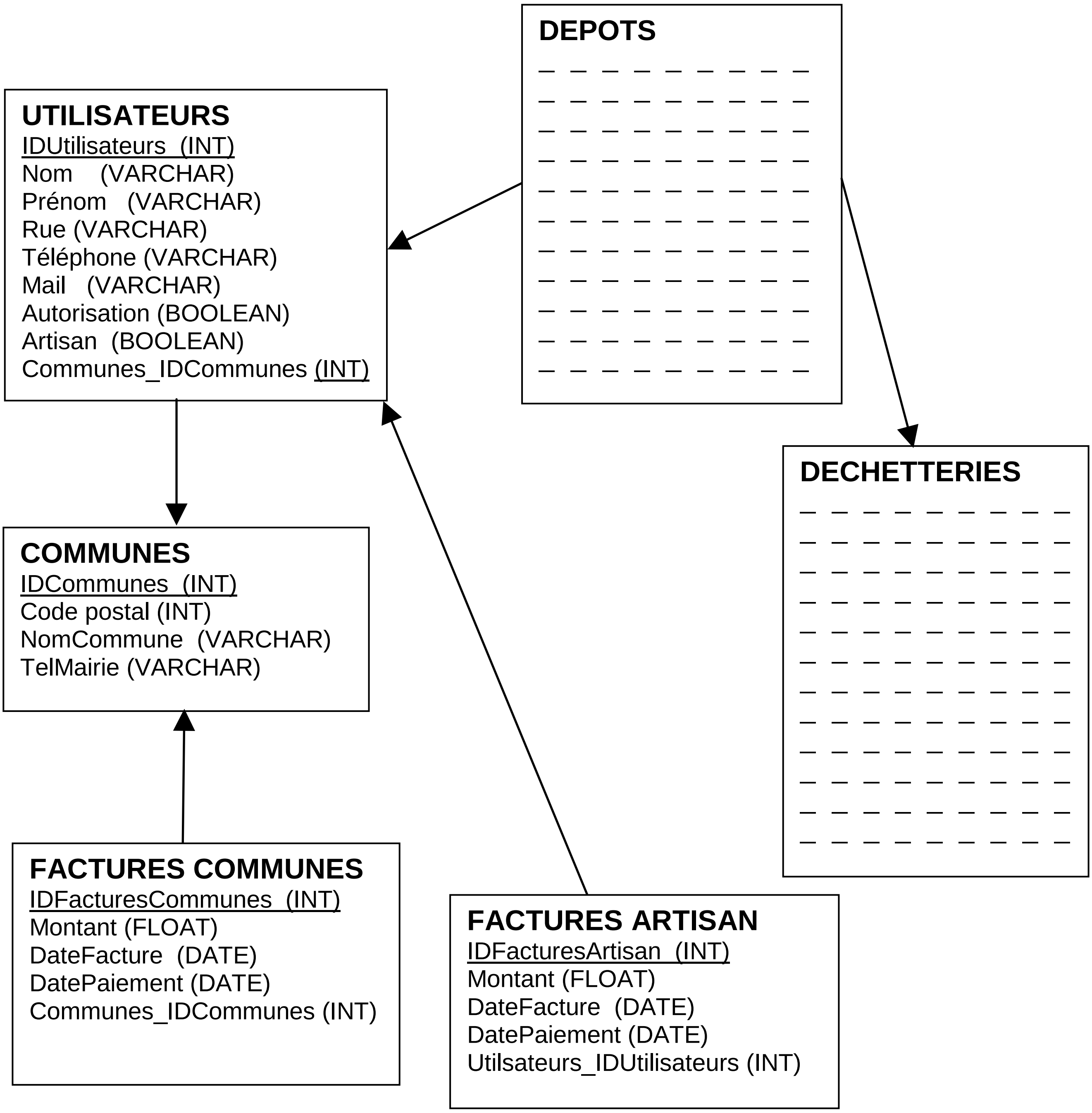
Question 18 en Scilab

Question 19

Question 20

Question 21

Schéma de la base de données à compléter



Identifiant(s)

_ _

_ _

Clé(s) étrangère(s)

_ _

_ _

Question 22

Donner la requête SQL :

Donner le résultat :

Question 23

Condition(s) :

Requête SQL :

Question 24

Requête SQL :

Résultat :

Question 25

Requête SQL **:**